ABSTRACT

DATA BROADCASTING AND REDUCTION, PREFIX COMPUTATION, AND SORTING ON REDUCED HYPERCUBE (RH) PARALLEL COMPUTERS

by
Arup Mukherjee

The binary hypercube parallel computer has been very popular due to its rich interconnection structure and small average internode distance which allow the efficient embedding of frequently used topologies. Communication patterns of many parallel algorithms also match the hypercube topology. The hypercube has high VLSI complexity. however. due to the logarithmic increase in the number of connections to each node with the increase in the number of dimensions of the hypercube. The reduced hypercube (*RH*) interconnection network. which is obtained by a uniform reduction in the number of links for each hypercube node. yields lower-complexity interconnection networks when compared to hypercubes with the same number of nodes. It has been shown elsewhere that the *RH* interconnection network achieves performance comparable to that of the hypercube. at lower hardware cost. The reduced VLSI complexity of the *RH* also permits the construction of larger systems. thus. making the *RH* suitable for massively parallel processing. This thesis proposes algorithms for data broadcasting and reduction. prefix computation. and sorting on the *RH* parallel computer. All these operations are fundamental to many parallel algorithms. A worst case analysis of each algorithm is given and compared with equivalent algorithms for the regular hypercube. It is shown that the proposed algorithms for the *RH* yield performance comparable to that for the regular hypercube.

DATA BROADCASTING AND REDUCTION, PREFIX COMPUTATION, AND SORTING ON REDUCED HYPERCUBE (RH) PARALLEL COMPUTERS

by
Arup Mukherjee

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

October 1994

APPROVAL PAGE

DATA BROADCASTING AND REDUCTION, PREFIX COMPUTATION, AND SORTING ON REDUCED HYPERCUBE (RH) PARALLEL COMPUTERS

Arup Mukherjee

BIOGRAPHICAL SKETCH

Author: Arup Mukherjee

Degree: Master of Science in Electrical Engineering

Date: October 1994

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1994

- Bachelor of Science in Electrical Engineering,
  University of Roorkee, Roorkee, India, 1991

Major: Electrical Engineering

This thesis is dedicated to
my parents Ajoy and Dipti and my brother Arjun

ACKNOWLEDGMENT

I would like to thank Prof. Ziavras for his guidance, encouragement and great insight during my work on this thesis.

Special thanks to Professors John D. Carpinelli and Edwin Hou for serving as members of the committee and for their perusal of my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Importance of Massively Parallel Processing Systems

Parallel processing in recent years has been making great strides in many areas of computer application. Parallel processing has made it possible to address many applications that were until recently beyond the capability of conventional computing. Massively parallel processors (MPP) are thought to be the most likely technology to achieve teraflops computational power. MPPs are large scale multiprocessors with thousands of nodes connected in a network. Each node has its own processor, local memory, and other peripheral devices. The way the nodes are connected varies widely. In a direct connected network architecture, each node has a direct connection to some other nodes. Direct connected multicomputers have become a popular architecture due to their support of scalability. As the number of nodes in the system increases, so does the processing capability, communication bandwidth, and memory bandwidth. The goal is to have teraflops performance by the end of this decade. Such tremendous computing power is needed in various fields, like aerodynamics, astrophysics, biology, and nuclear physics for detailed simulations.

1.2 The Hypercube Topology

The objective in building a commercial MPP system is to have a general purpose architecture on which a number of different types of problems can be solved. One such general purpose topology is the hypercube which has been widely researched. It is also called the direct binary n-cube. A n-dimensional hypercube has 2^n nodes. If unique consecutive binary n-bit addresses are assigned to its nodes, then nodes whose addresses differ in only one bit have a direct link between them. A hypercube can be constructed recursively as follows: a $(n+1)$-dimensional hypercube is constructed by connecting the corresponding processors of two n-dimensional hypercubes. The hypercube has been a successful architecture due to the following properties:

1. *Low diameter* in large systems. The diameter of an interconnection network is defined as the maximum distance between all pairs of nodes. For a n-dimensional hypercube, the diameter is n.

2. It has a *general purpose* topology. The hypercube can emulate widely used structures very efficiently. There has been significant research in this area. Algorithms for mapping rectangular meshes have been proposed among others by Chan and Saad [14], and Johnsson [16]. Binary tree mappings were proposed by Wu [15], Deshpande and Jenevin [3], Ho and Johnson [16], and Leighton [5] among others. Algorithms for mapping pyramids have been proposed by Chan and Saad [14], Lai and White [11], and Ziavras and Siddiqui [12], among others.

3. It has a *fault tolerant* robust architecture due to its high degree of connectivity.

Several commercial hypercube computers have been manufactured. The Thinking Machines CM-2, the NCUBE, and the Intel iPSC are the most important among them. The CM-2 has up to 65,536 PE's which are simple 1-bit processors. The other two machines have a smaller number (up to 1,024) of powerful processors. An Intel iPSC/1 node has an Intel 80286 processor, with 512 KB of memory. Each node can be expanded to add floating point accelerators, extra memory, or I/O devices. Ethernet chips are used to implement communication channels between nodes. Another channel from each node is used to implement connection back to a host. This host processor is called the Cube Manager. The Cube Manager is connected to the processors in the cube by a broadcast bus for global communication, I/O, and control.

Systems that have a pure hypercube network have two major drawbacks: (1) the size of the system has to be an integer power of two; and (2) the number of communication ports and channels per processor increase logarithmically with the increase in the total number of processors in the system which increases dramatically the total number of communication channels [1]. This VLSI constraint prevents building powerful, massively parallel hypercube systems.

1.2.1 Variations of the Hypercube Topology

The high VLSI complexity of the hypercube has led many researchers to look into *hypercube-like* topologies with lower VLSI complexity. This section takes a look at some existing hypercube variations. The reduced hypercube is another of these variations and is described in the next section. The *cube connected cycles* CCC(n) [7] is obtained from the n-dimensional hypercube by substituting a ring with n nodes for each node in the hypercube. Each node in a ring then implements a distinct connection in one of the n dimensions. The advantage of the CCC(n) is that the node connectivity is always 3, independently of the value of n.

The *incomplete hypercube* [18] is another important variation of the hypercube. An incomplete hypercube is constructed by connecting together two complete hypercubes of different sizes. The major disadvantage of the incomplete hypercube is that a large number of communication ports may be wasted and as a consequence a significant portion of the system's cost may be spent for unused resources. For example, an incomplete hypercube with 1,280 processors can be constructed from two complete hypercubes composed of 1,024 and 256 processors, respectively. The interconnection of two complete hypercubes requires a number of communication ports per processor equal to 11 and 9, respectively for the two constituent hypercubes (this is in contrast to 10 and 8, respectively, for the corresponding conventional hypercubes). The total number of unused communication ports in this system is equal to 768 (i.e 1,024 - 256), assuming that all the nodes of the smaller hypercube are used. The VLSI complexity of the incomplete hypercube is also not drastically reduced for parts of the system, as was the goal. Another variation of the hypercube is the *hierarchial cubic network (HCN)* [4] which also uses the hypercube as the basic building block. A number of other variations of the hypercube have been proposed in the literature, but they do not reduce its VLSI complexity rather they sometimes increase it in order to achieve better topological properties.

1.3 The Reduced Hypercube

The reduced hypercube (RH) interconnection network has been proposed by Ziavras [2] in order to reduce the large VLSI complexity of the regular hypercube and, thus, facilitate the construction of larger systems. Although a RH can be viewed as a hierarchical structure with several levels, only the properties of structures with two levels were studied extensively. The algorithms developed in this thesis also assume RH's with only two levels. A RH is formed by uniformly removing several edges from the hypercube with the same number of nodes. The *reduced hypercube* $RH(k,n)$ contains a total of N nodes, where $N = 2^{k+2^n}$, with $k \geq n$ and $n > 0$. Each node of the $RH(k,n)$ is attached to $k+1$ bidirectional links. In a regular hypercube with the same number of nodes, each node is attached to $k+2^n$ bidirectional links. Therefore, each node in the N-node RH has $k+2^n-(k+1)$, or 2^n-1 links less than each node in the N-node regular hypercube.

The N-node $RH(k,n)$ is constructed from the N-node regular hypecube by uniformly removing $2^n - 1$ links from each of its nodes. To accomplish this, the $(k+2^n)$-bit addresses of hypercube nodes are first partitioned into two fields; the 0^{th} and 1^{st} fields, as follows. The 0^{th} field contains the k least significant bits of the $(k+2^n)$-bit node address. This field represents the address of the node within a complete k-cube, which will be referred to as a *building block (BB)*. The 1^{st} field contains the 2^n most significant bits of the node address. It represents the address of the BB that contains the node. In addition, a subfield is identified in the 0^{th} field. the 0^{th} *subfield*. It contains the n most significant bits of the k-bit 0^{th} field. It represents the address of a $(k-n)$-dimensional subcube, which will be referred to as a *subblock (SB)*, within the k-cube BB that contains the node. For simplicity let the term $k+2^n$ be denoted by ν from now on.

In order to reduce the ν-cube into the $RH(k,n)$, out of the ν (bidirectional) links of each hypercube node the following two sets are kept, leaving $k+1$ links to each node.

Set 1: The k links of the ν-cube that traverse the k lowest dimensions (i.e., dimensions 0 through $k-1$) and connect the referenced node with k distinct nodes are kept. As a result, a complete k-dimensional building block (BB) that includes the referenced node is kept.

Set 2: This set contains only one link which is also present in the original ν-cube. This link is the one which connects directly the referenced node with the node whose address differs only in the m^{th} bit of the 1^{st} field, where m is the decimal value in the 0^{th} subfield and $0 \leq m \leq 2^n - 1$.

The resultant $RH(k,n)$ contains 2^{2^n} k-cube BB's. It can also be viewed as a 2^n-cube of k-cube BB's. A BB address forms the 2^n most significant bits (i.e., the 1^{st} field) of the ν-bit addresses for contained nodes. Each BB is divided into 2^n subblocks (SB's); each SB is a $(k-n)$-cube. Connections between pairs of SB's in different BB's are as follows: A node in a particular SB of a particular BB is connected to the node with the same 0^{th} field address which belongs to the BB whose 2^n-bit address differs only in the m^{th} bit, where m is the value in the 0^{th} subfield of the former node. It was shown in [2], that the RH can emulate simultaneously, with dilation equal to one, several cube-connected cycles networks.

Figure 1.1 shows the structure of the $RH(3,1)$. There are 2^n, that is 2^1, SB's in each BB. Each BB is a complete 3-cube, since $k = 3$. BB addresses appear above each BB. BB addresses have two bits. SB addresses have one bit and appear inside the BB box. Links between nodes in different BB's are shown by dashed lines.

Figure 1.2 shows the structure of the $RH(k,2)$ where the large squares represent the k-cube building blocks. The numbers above the squares represent in decimal the BB addresses and the numbers within the quadrants of large squares are the SB addresses in decimal. To keep the figure simple, the nodes within the square are not shown. Each line between BB's represents 2^{k-2} bidirectional communication channels; this is also the number of nodes in each SB. It is implied that each node in a SB is connected to the node with the same 0^{th} field address in the SB where the connection line leads.

1.3.1 Hypercube Emulation on the RH

The RH is equivalent to a hypercube with a smaller number of links per node. Therefore, the performance of the topology may degrade for algorithms designed explicitly for the hypercube. The algorithms given in this thesis are not pure hypercube algorithms. They use the hypercube structure within the BB's and then use the communication links between the BB's. The emulation of the hypercube by the RH has been investigated in [2] and the most important results are presented here.

The dilation of edges associated with the chosen hypercube mapping must be found for evaluation of the performance. The dilation measures the increase in communication steps to reach a neighboring node, as compared to the hypercube. Let the regular ν-dimensional hypercube and the target $RH(k,n)$ contain the same number of nodes; that is 2^ν, where $\nu = k + 2^n$. Assume that nodes from the regular

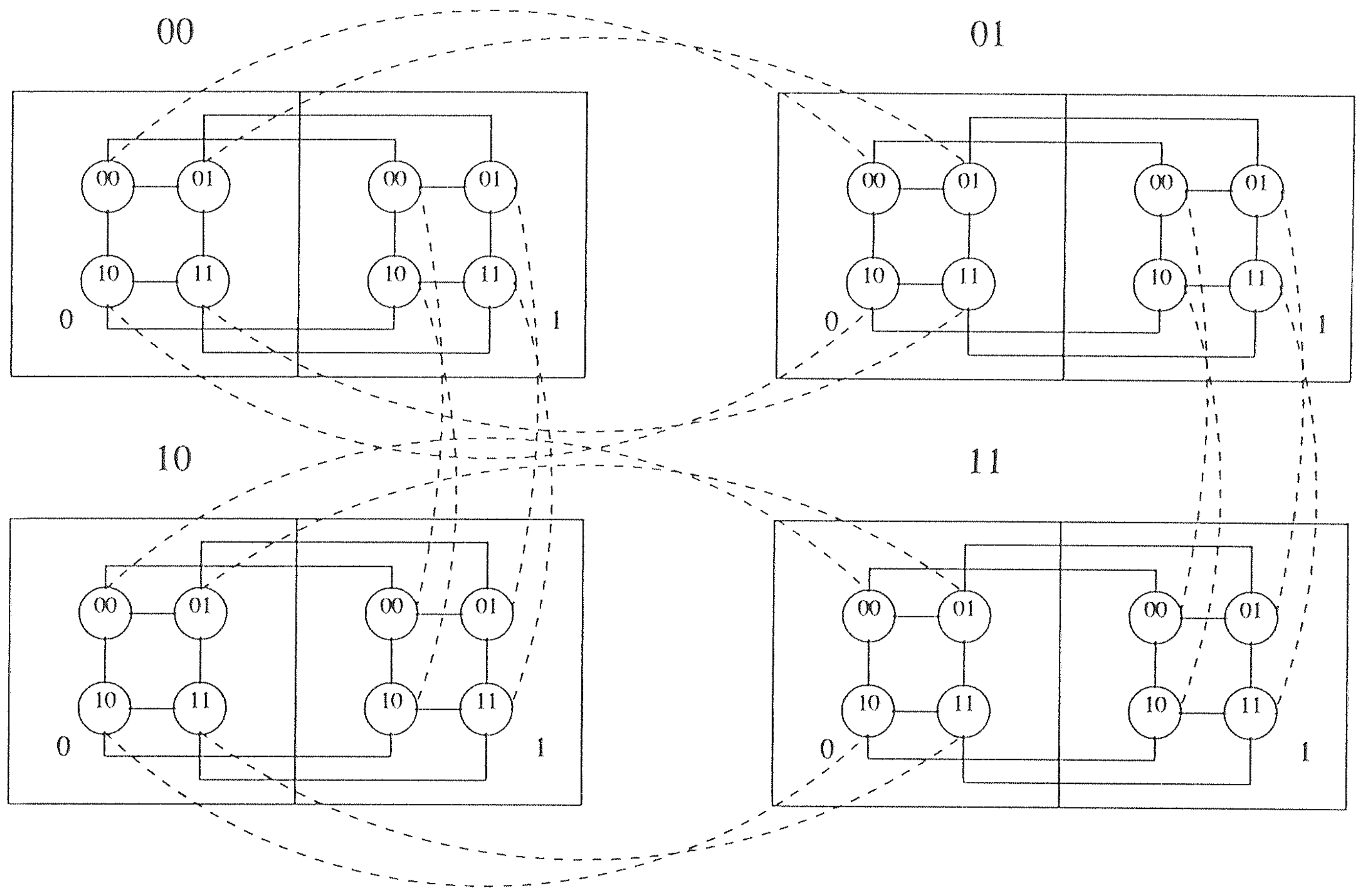


Figure 1.1 The structure of the RH(3,1)

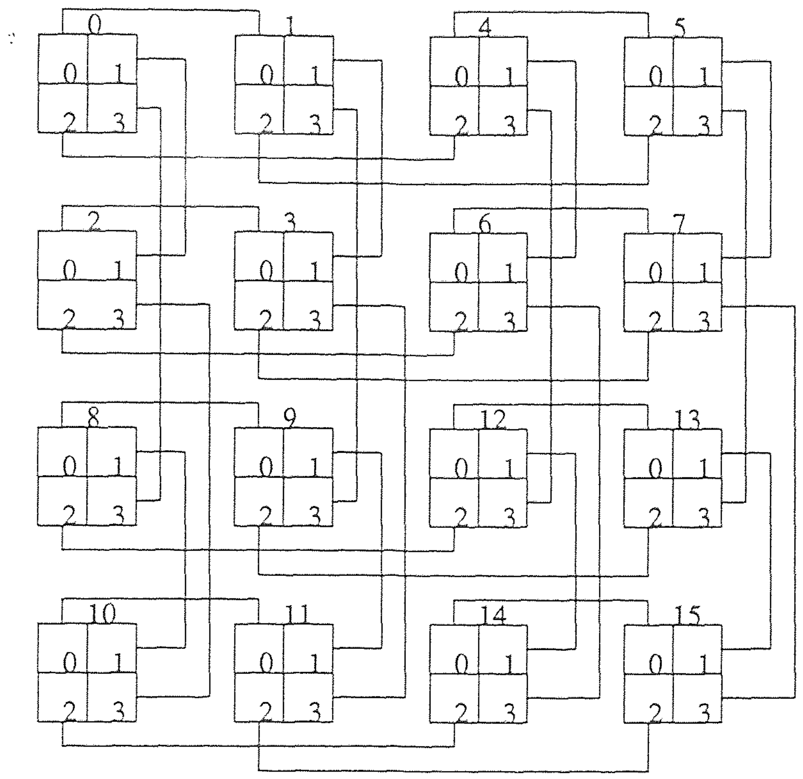


Figure 1.2 The structure of the $RH(k,2)$

hypercube are mapped to nodes of the RH with the same address. The following theorem [2] presents the resultant dilation of edges.

Theorem: For the emulation of the ν-dimensional hypercube on the reduced hypercube $RH(k,n)$ with the same number of nodes, the dilations of edges incident to a single node of the hypercube are: 1 for $k+1$ of them and $2p+1$ for $\begin{pmatrix} n \\ p \end{pmatrix}$ of them, where $p = 1, 2, \ldots, n$, and $\begin{pmatrix} n \\ p \end{pmatrix}$ represents the number of distinct p -combinations of n items.

Example: The dilations of the edges incident to a single node of the $RH(5,2)$ for the emulation of the 9-dimensional hypercube are 1, 3 and 5 for 6, 2 and 1 edge, respectively. Similarly, the dilations of the edges incident to a single node for the emulation of the 16-dimensional hypercube on the $RH(8,3)$ are 1, 3, 5 and 7 for 9, 3, 3 and 1 edge, respectively.

The maximum and average dilations are two other important metrics for hypercube emulation on the RH. The following two corollaries provide the means for their calculation [2].

Corollary 1: The maximum dilation of edges for hypercube emulation on the $RH(k, n)$ is equal to $2n + 1$.

Corollary 2: The average dilation of edges for hypercube emulation on the $RH(k, n)$ is equal to

$$\frac{k + (n + 1)2^n}{k + 2^n}$$

The average dilation of edges for the last two examples is 1.88 and 2.5, respectively. The average dilation of edges has been shown in [2] to be relatively small in practical cases. So, there is a small performance degradation for the implementation of hypercube algorithms on RH's. The effect of dilation is reduced significantly from left to right for the set of four well-known packet switching techniques: store-and-forward, virtual cut-through, circuit switching, and wormhole routing. The ring, the torus, and the binary tree have been mapped efficiently on the RH [17]. These topologies are very frequently used in parallel algorithms.

We assume a MIMD message passing multicomputer environment for all the algorithms developed in this thesis. In this model each node has its own processor and memory. Since they do not physically share memory, nodes communicate by passing messages through the network. A message is often broken into packets. A packet is the smallest unit of communication that contains routing and sequencing information which is carried in the packet header. Neighboring nodes send packets to one another directly but nodes which are not directly connected rely on intermediate nodes in the network to relay packets from source to destination. Most systems now have a dedicated router in each node to handle communication related tasks, to allow overlapped computation, and communication within each node. The programmer of a multiprocessor invokes various communication system calls to achieve interprocessor communication.

CHAPTER 2

BROADCASTING ON THE REDUCED HYPERCUBE (RH)

Broadcasting is a very common operation in parallel algorithms. Initially one processor has a data element that needs to be broadcast. At the end of the broadcasting procedure, there is a copy of the data element in every processor in the system. Broadcasting is used in several parallel algorithms including matrix-vector multiplications, Gaussian elimination, shortest paths, and vector inner product. The following section gives the broadcasting procedure for the $RH(k,n)$ for the special case where $k = n$. In the subsequent section the broadcasting procedure will be generalized to include the $RH(k,n)$, for $k > n$. The binary tree is the basic structure which is used for the broadcasting procedure.

2.1 Broadcasting on the Reduced Hypercube RH(n,n)

In the first phase of the algorithm the 2^n most significant bits of each node's address are used to map a (complete) binary tree with 2^n levels onto the 2^n-dimensional hypercube of BB's. The binary tree is double-rooted (using a spacer node) to utilize all the BB's in one-to-one mapping [3]. For example, Figure 2.1 shows the double-rooted binary tree of depth 2 that utilizes all the nodes in the 3-cube. Assume that the index of the LSB in the node address is 0, so that MSB (most-significant bit) refers to the bit with offset $2^n + k - 1$. Only the 2^n most significant bits of node addresses are considered in the first phase. Each virtual node in the mapping is actually an n-dimensional hypercube BB, therefore one of each BB's internal nodes will receive the broadcast value from its parent (except for the root) and up to two other internal nodes will have to transmit the received value to their children located in two other BB's.

In the second phase each node within a BB determines whether it is the Node-of-Entry (NOE) or a Node-of-Exit (NOX) for the implementation of connections to parent and child BB's. An algorithm for broadcasting a value from the NOE to all other nodes in a BB must be also introduced. In the third and final phase, without loss of generality, the value is broadcast starting from the node with address 0 in the root BB in the tree of BB's. The aforementioned phases of the algorithm are described in detail in the remaining subsections.

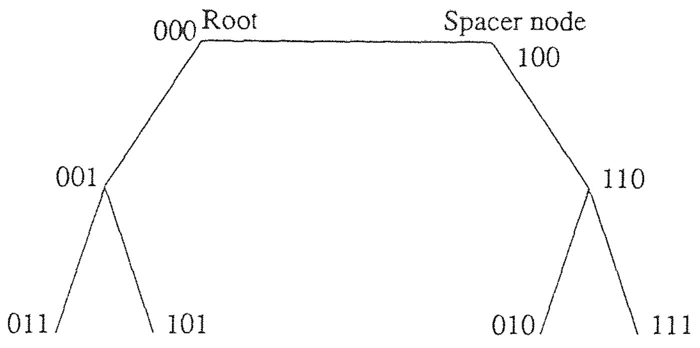


Figure 2.1 Double-rooted binary tree with three levels

2.1.1 Phase I: Setting up the Binary Tree Configuration of BB's

The 2^n-level binary tree of BB's is obtained by applying an algorithm that implements one-to-one mapping of a binary tree with 2^{2^n}-1 nodes onto the 2^n-dimensional hypercube [3] of BB's. This phase of the algorithm starts by setting up initially 2^{2^n-3} three-level double-rooted binary trees having a predetermined configuration. That is, every BB becomes a member of a three-level tree; its position in the tree is determined by the values of its bits 0,1 and 2 in its 2^n-bit address. The algorithm given below is run by all 2^{2^n+n} nodes in the $RH(n,n)$ using the 2^n most significant bits of their addresses. Two transformations of the 2^n-bit BB addresses are used in the algorithm [3]. Tables 2.1 and 2.2 give the transformations. The transformations satisfy the following two properties:

1. Nodes with distinct addresses map to distinct target nodes.

2. If two nodes are neighbors and thus have addresses differing in only one bit position, their new addresses after the transformations also differ in only one bit position. Thus, neighborhood between the two nodes is preserved for optimal mapping of the three-level tree.

Table 2.1 Transformation FT3

$x_i x_j x_k$	$y_i y_j y_k$
000	100
001	000
010	101
011	001
100	110
101	010
110	111
111	011

Table 2.2 Transformation BT3

$x_i x_j x_k$	$y_i y_j y_k$
000	001
001	101
010	000
011	100
100	011
101	111
110	010
111	110

At each successive iteration of the algorithm, trees are merged to form larger trees until eventually a binary tree is formed that contains all BB's. The merging of two equal-sized trees requires a spacer node. By the introduction of a single two-degree node as the child of its root, and thereby stretching (or equivalently double rooting) it, the tree can be made to utilize a hypercube completely. The extra node so introduced is used only for communication between the root and one of its children, and is called the spacer node. At the end of this phase of the algorithm each node in each BB knows which BB (if any) is its parent, and which BB's (if any) are its children and also their virtual and physical addresses. The tree setup algorithm adapted from [3] follows.

The algorithm is run by all PE's, with each PE assuming that it is the only one in the corresponding BB; these PE's will also be called virtual nodes. Each uses the following variables:

- *current-port:* Every virtual node in the hypercube formed by the BB's has 2^n ports, each one corresponding to a bit position in its 2^n-bit address. This variable keeps a running pointer to the bit position currently being considered.
- *physical-id:* Original BB address of the virtual node.
- *current-id:* The virtual node BB address during the current iteration.
- *port-relation(1..2^n):* An array of values specifying the current active connections of the virtual node. All are initialized to "null" (inactive).

  All possible values assigned to the *port-relation(i)* variable are:

  null: No active connection.

  p: Connection to parent virtual node.

  c: Connection to child virtual node.

The following is the tree setup algorithm of the BB's:

```
for-all virtual nodes do
begin
  current-id = physical-id;
    /*set up 2^{2^n-3}, 3-level trees*/
        case current-id(bits: 2..0) of
          0:  port-relation(0) = port-relation(2) = c;
          1:  port-relation(0) = p;
              port-relation(1) = port-relation(2) = c;
          2:  port-relation(2) = p;
          3:  port-relation(1) = p;
          4:  port-relation(1) = c;
              port-relation(2) = p;
          5:  port-relation(2) = p;
          6:  port-relation(0) = port-relation(2) = c;
             port-relation(1) = p;
          7:  port-relation(0) = p;
        end-case
for current-port = 3 to 2^n-1 do
          begin
          /*Form larger trees iteratively*/
          if (current-id(current-port)=1) then
           begin
           Apply FT3 to current-id's bits 2,1 and 0;
           /* This transformation is given in Table 1 */
           end
          if (Bits 3 through current-port-1 are 0) then
           begin
            case current-id(bits: current-port,2,1,0) of
              0: port-relation(2)= p;
                 port-relation(current-port)=c;
              4: port-relation(2)= c;
                 port-relation(current-port)= c;
                 port-relation(1)= null;
              6: port-relation(1)= null;
                 port-relation(current-port)= p;
```

```
                8: port-relation(2)= null;
                   port-relation(current-port)= p;
               12: port-relation(2)= null;
                   port-relation(current-port)= p;
               14: port-relation(current-port)= c;
            end-case
          end
          Apply BT3 to bits current-port, 2 and 0 of current-id;
          /* This transformation is given in Table 2 */
        end
end
```

Figure 2.2 shows the case of merging two k-level binary trees with spacer nodes in two k-cubes to form a *(k+1)*-level binary tree in a *(k+1)*-cube [3]. The steps are as follows:

1. Apply the FT3 transformation to the nodes of the duplicate mapping using bit-2 as x_i, bit-1 as x_j and bit-0 as x_k to obtain the mapping given in Figure 2.2(b).

2. Form a $(k+1)$-cube by connecting the nodes with like addresses in the two k-cubes. Append a 0 to the left of addresses in the original k-cube and a 1 to the left of addresses in the duplicate k-cube.

3. Remove links 0S100-0S110 and 1S100-1S000, and attach links 0S000-1S000, 0S110-1S110 and 0S100-1S100 as shown in Figure 2.2(c) to obtain Figure 2.2(d).

4. Apply the BT3 transformation to the nodes of the $(k+1)$-cube to obtain a $(k+1)$-level double-rooted binary tree rooted at 0S000 (it is 0S100 before the transformation). In applying the BT3, use the most significant bit as x_i, the third least significant bit as x_j, and the least significant bit as x_k. Replace 0S by S` to obtain a structure similar to the base structure we started with. The resultant mapping is shown in Figure 2.2(e).

Because of the binary tree mapping. each BB corresponds to one of the following cases:

1. It has two children and no parent. This is the root BB.
2. It has a parent and a single child. This is the second root or spacer BB.
3. It has a parent and two children. These are all intermediate BB's, excluding the spacer BB.
4. It has a parent and no children. These are the leaf BB's.

All nodes within a BB produce the same parent and/or children BB addresses in Phase I.

2.1.2 Phase II: Determining the Nodes-of-Entry and Nodes-of-Exit

Each node then determines whether it is directly connected to a parent or a child BB. It does this by comparing its BB address with the address of parent (if any) and child (if any) BB's computed in Phase I. If such a comparison shows a difference in a single bit with offset equal to the value stored in the 0^{th} subfield of the nodes address. the node knows it is directly connected to the corresponding parent or child BB. Each node which is directly connected to a parent BB marks itself as Node-of-Entry (NOE). Each node which is directly connected to a child BB marks itself as Node-of-Exit (NOX). We must remind here that each SB in the $RH(n, n)$ contains a single node. so there is no ambiguity with regards to the chosen node. Each BB will have up to one NOE node and up to two NOX nodes according to the classification presented in subsection 2.1.1. Each BB then internally maps a binary tree with the NOE node as the root using the algorithm [3] presented for the first phase and assuming an n-cube as the target system.

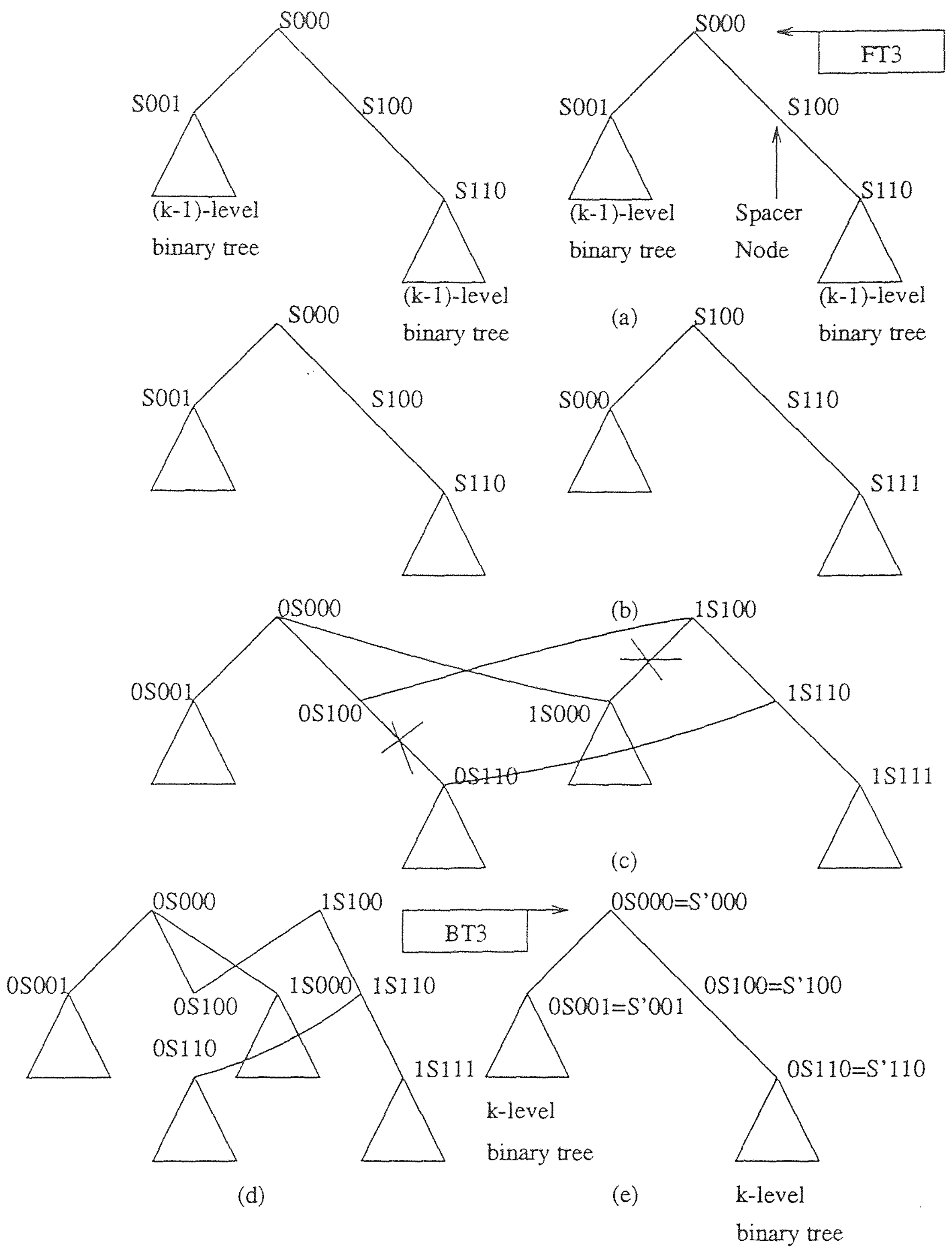


Figure 2.2 Binary tree merging in the hypercube

2.1.3 Phase III: Broadcasting the Value to all Nodes

Assume, without loss of generality, that the value to be broadcast is stored in the node with address 0 in the root BB with address 0. The value is then broadcast to the BB's using the binary tree of BB's. In each BB the NOE node receives the value first and passes on the value to its children following a binary tree mapping for the n-cube BB. If a node that receives the value is a NOX, it passes on the value to the neighbor in the next level of the binary tree of BB's, and also passes on the value within the same BB using the internal binary tree mapping. If an intermediate node is not a NOX, it just passes on the value to its two children in the same BB using the internal binary tree mapping. To broadcast a value from a node other than 0 in BB 0, simple transformation of addresses is needed because of the symmetry in the n-cube BB's and in the 2^n-cube of BB's.

2.1.4 Analysis of the Algorithm

Phase I:

According to [3] the tree setup algorithm requires time $O(2^n)$ for the 2^n-cube of BB's.

Phase II:

It takes time $O(2^n)$ for each node to determine whether it is NOE, a NOX, or neither, because 2^n bits must be checked. The mapping of a binary tree onto the n-cube BB consumes time $O(n)$. So this phase takes time $O(2^n)$.

Phase III:

Broadcasting on the 2^n-cube of BB's requires time $O(2^n)$ because of the binary tree mapping. Broadcasting within a single n-cube BB requires time $O(n)$ because of the binary tree mapping. Therefore, this phase takes time $O(n2^n)$.

Therefore, the overall time complexity of the algorithm is $O(n2^n)$. In contrast, broadcasting on the $(2^n + n)$-dimensional hypercube with the same number of nodes requires time $O(2^n + n)$ or $O(2^n)$. However, in practical cases the value of n is small, that is 1, 2, 3, or 4 [2], therefore, broadcasting on the two systems requires comparable amounts of time.

2.2 Broadcasting on the Reduced Hypercube RH(k,n), where k>n

This subsection generalizes the broadcasting procedure given in the previous subsection for the $RH(n,n)$ to make it applicable to the $RH(k,n)$, where $k > n$. It has been mentioned in [2] that for $k > n$ the $RH(k,n)$ is viewed as 2^{k-n} $RH(n,n)$'s where all nodes with the same address in the 2^{k-n} distinct $RH(n,n)$'s are connected to form a $(k-n)$-dimensional hypercube. The nodes' addresses in the latter hypercube become the least significant $k-n$ bits of the nodes' addresses in the $RH(k,n)$. This property will be used in this section in order to follow basically the algorithm of section 3.1.

Without loss of generality, assume broadcasting from the processor with address 0. All nodes with zeros in the $2^n + n$ most significant bits of their address participate in the first phase of the algorithm. In this phase a $(k-n)$-level binary tree with a spacer node is mapped to a $(k-n)$-cube in BB 0. This hypercube contains all nodes that have all zeros in the $2^n + n$ most significant bits of their address. Broadcasting is then carried out in this binary tree within BB 0, starting from the node with address 0. Ignoring the $k-n$ least significant bits of node addresses, broadcasting is then implemented independently within the distinct 2^{k-n} $RH(n,n)$'s. Broadcasting begins with that node of each $RH(n,n)$ whose all $2^n + n$ most significant bits in the address are zeros; this broadcasting follows the procedure given in section 2.1.

2.2.1 Analysis of the Algorithm

The broadcast of the value within the $(k-n)$-cube of BB 0 requires time $O(k-n)$. The parallel broadcast of the value within the distinct $RH(n,n)$'s requires time

$O(n2^n)$, as given in section 2.1.4. Therefore, the overall time complexity of the broadcast algorithm for the $RH(k, n)$ is $O((k - n) + n2^n)$ or $O(k + n2^n)$. In contrast, broadcasting on the $(2^n + k)$-dimensional hypercube with the same number of nodes requires time $O(2^n + k)$. However, in practical cases the value of n is small, that is 1, 2, 3 or 4, therefore, broadcasting on these two systems requires comparable amounts of time.

CHAPTER 3

REDUCTION OPERATION ON THE REDUCED HYPERCUBE (RH)

Data reduction is an operation where an associative operator must be applied to values stored one per processor, in order to produce a single result. The common associative operators are logical OR, logical AND, maximum, minimum, and add. For example, consider the operation in which one processor in the reduced hypercube wants to know the sum of the values stored in all the processors including itself. Reduction often facilitates barrier synchronization on message-passing parallel computers. The concept of barrier synchronization is that a set of processes in execution cross a "barrier" as an atomic action: it means that after all processes have reached the barrier, all traverse it at once [20]. Barrier synchronization is useful for separating different phases of a concurrent algorithm.

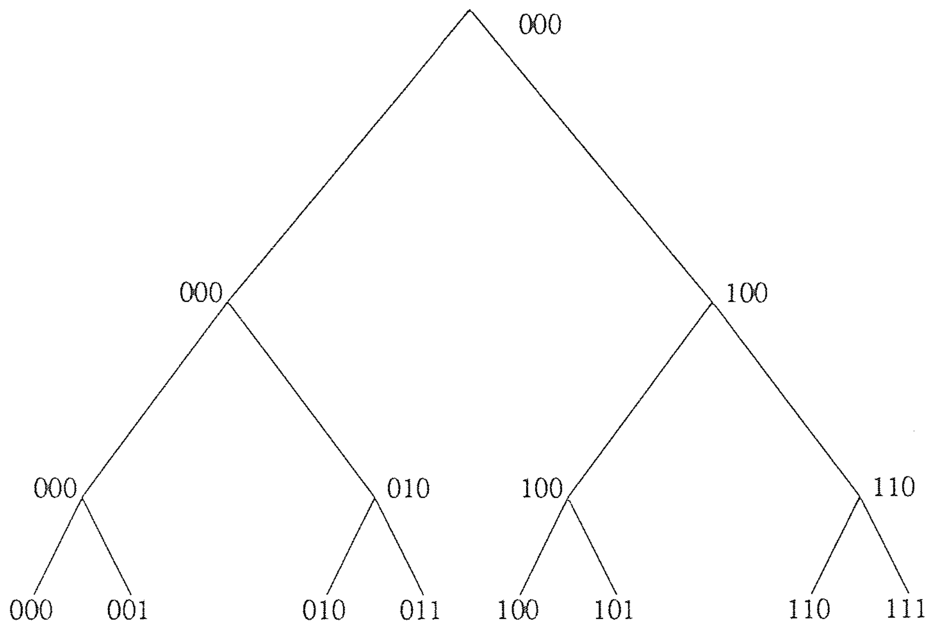
3.1 Data Reduction Algorithm

Many-to-one mapping of a binary tree is very suitable for the implementation of the reduction operation on a hypercube. A binary tree of height d can be optimally mapped in a many-to-one manner onto a hypercube with 2^d nodes as follows [8]:

1. The root of the tree is mapped onto any hypercube node.

2. For each node i at depth j (the root is at depth 0), the left child of i is mapped to the hypercube node i, and the right child of i is mapped to the hypercube node whose address is obtained by inverting bit p - j + 1 of node i 's address, where p is the offset of the most significant bit. Nodes from any single level of the binary tree are mapped to distinct hypercube nodes.

Figure 3.1 shows the mapping of a tree of height 3 onto a hypercube of dimension 3, assuming that the root has address 0. Since we determine the right child of a node by complementing one bit of its address, there is an edge in the hypercube that directly connects these two nodes. We also see that the leaves are consecutively numbered.



The address of hypercube nodes is shown.

Figure 3.1 Many-to-one mapping of a binary tree of depth 3 onto a hypercube of dimension 3

The data reduction algorithm for the RH proceeds as follows. A binary tree is first mapped onto each k-cube BB, according to this many-to-one manner. Each node at depth k-1 does a reduction operation with its right child which is a leaf. Then each node at depth k-2 does a reduction operation with its right child, and so on till we reach the root, which is chosen to be the node 0 in the BB. Each BB now has a node with the result of the reduction operation for the BB. The 2^n most significant bits of node addresses are then used to map in a many-to-one manner a (2^n+1)-level binary tree onto the 2^n-cube of BB's, using the algorithm given above. At most n hops are required within a BB to go from the node which has the reduction operation value for that BB to the node (whose $k - n$ least significant bits are zeroes) which implements a connection to its parent BB in the binary tree of BB's, and then at most another n hops to go from the latter node to the node which has the reduction operation value for the parent BB. This node then performs the reduction operation. If it is the right child of its parent, it passes on the value to the parent BB as indicated above. So, there is a dilation of at most $2n + 1$. At the end of this stage, the node with address 0 in the $RH(k, n)$ will have the final reduction operation value.

3.2 Analysis of the Algorithm

It takes d steps for a reduction operation to be done on a tree of height d. The binary tree of BB's has height 2^n for the hypercube. The reduction operation within BB's requires time $O(k)$. The reduction operation between pairs of BB's requires time $O(n)$. Since the reduction operation among BB's requires time $O(n2^n)$, the total time required is $O(k + n2^n)$. The reduction algorithm for the hypercube with the same number of nodes has a time complexity of $O(2^n + k)$ because a $(2^n + k + 1)$-level binary tree will be mapped in a many-to-one manner. Since the value of n is small in practical cases, it takes comparable amounts of time for the implementation of the reduction operation on the two systems.

CHAPTER 4

PREFIX OPERATION ON THE REDUCED HYPERCUBE (RH)

Prefix computation is commonly used in various parallel algorithms, including the evaluation of polynomials, ranking and packing problems, solution of linear recurrences, carry look-ahead addition, finding convex hulls of images, and scheduling problems. Given p numbers $n_0, n_1, \ldots, n_{p-1}$, the prefix computation problem is to compute $s_k = n_0 \oplus n_1 \oplus \ldots \oplus n_k$, for all k between 0 and $p-1$, where $\oplus$ denotes an associative operator. Initially n_k resides in the processor with address k, and at the end of the procedure the same processor holds s_k.

4.1 Phase I: Prefix Operation within BB's

Each processor in the BB maintains two parameters, namely $rslt$ and msg [8]. These parameters are initialized with the value n_i that the processor contains. k steps follow. In step i, each processor sends its msg parameter to its neighbor in dimension i, for $i = 0, 1, 2, \ldots, k-1$. Its new msg value is obtained by applying $\oplus$ to the old msg value and the one received. If the incoming value comes from a lower-addressed neighbor, then assign to $rslt$ the value obtained by applying $\oplus$ to the old $rslt$ value and the one received.

4.2 Phase II: Prefix Operation among BB's

In this phase the connections between the subblocks of different BB's are utilized for the prefix computation. The algorithm goes through several steps. In each step some BB's receive a prefix value from other BB's and pass on the value to the node with address $2^k - 1$ in the BB. This node applies the associative operator to the values it receives in order to combine the result at the end with the value it contained in the end of the first phase and send it to other BB's. It also keeps a copy of the result for the rest of the nodes in its BB. When it has received the prefix values from all preceding BB's, it broadcasts the result to all nodes in the BB which update their prefix values. The receiving BB's in all steps also follow this rule: if they receive the value from a BB which is labeled 2^m higher or lower than themselves, they pass on the value to the BB's which are labeled 2^0, 2^1, 2^2, ..., and 2^{m-1} lower than themselves.

In the first step each BB which has a one in bit position zero of its address receives the prefix value from the BB whose address differs (from its own address) only in that bit. The receiving BB's follow the rule outlined earlier. In the second step each BB which has ones in bit positions 1 and 0 receives the prefix value from the BB whose address differs only in bit position 1. So, in each step the prefix operation is carried out in one of the dimensions of the hypercube formed by the BB's. These steps can be generalized by the following loop:

```
for $i$=0 to $2^n - 1$ do
begin
Each $BB$ which has ones in bit positions 0, 1 , ..., $i$ of its address receives the prefix value from the $BB$ whose address differs only in bit position $i$. The receiving $BB$'s follow the rule outlined earlier.
end
```

Each of the steps above has substeps where the receiving BB's distribute the prefix value calculated up to that stage, as discussed earlier. Figure 4.1 shows the communication steps between BB's for the prefix operation in the $RH(k, 2)$ (see Figure 1.2). Each node in the figure represents a BB with four subblocks.

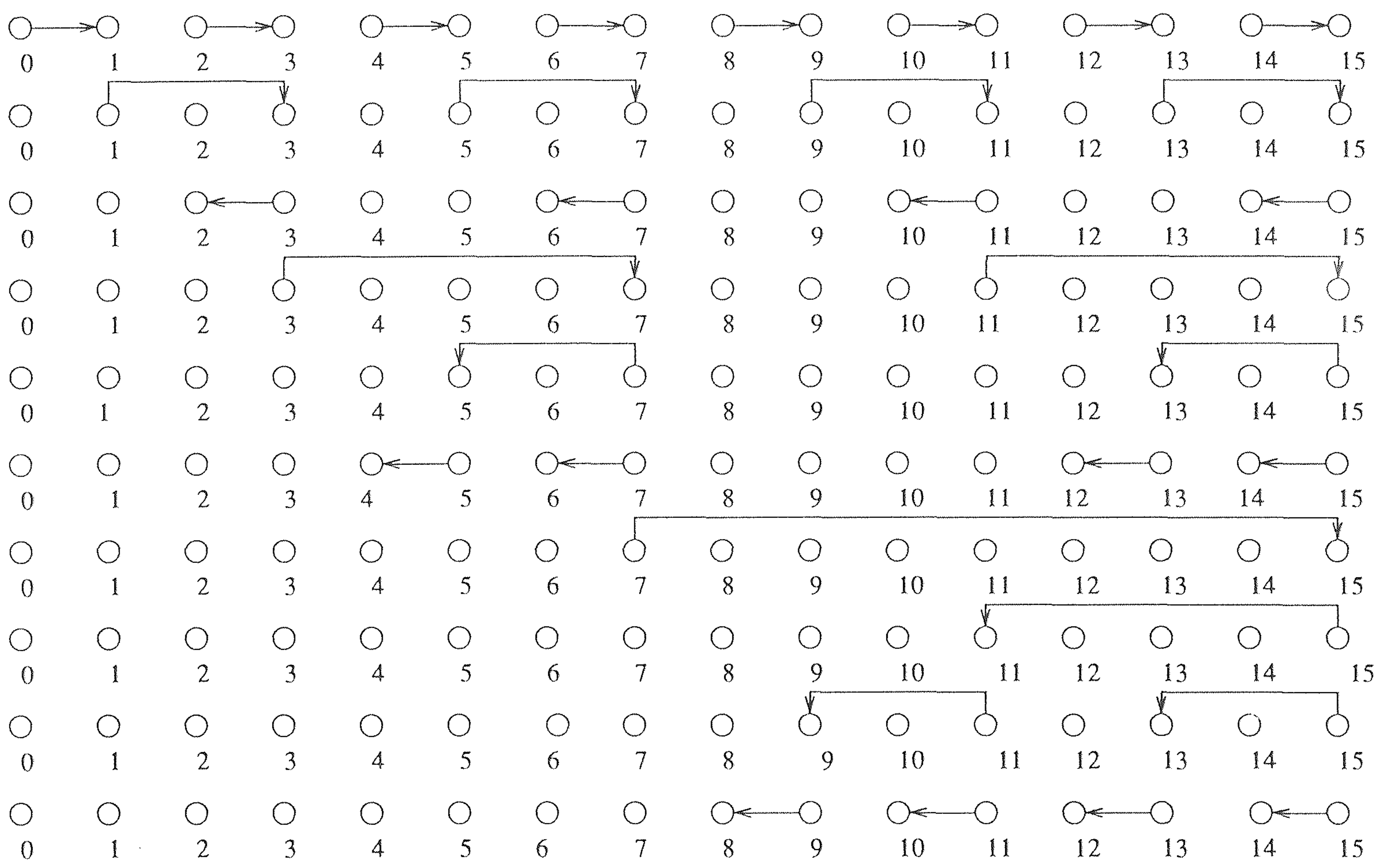


Figure 4.1 Prefix operation between BB's for the RH(k,2)

4.3 Analysis of the Algorithm

The first phase of the algorithm takes time $O(k)$. Each step in the second phase of the algorithm has substeps as shown in Figure 4.1. The total number of substeps is 1+2+3+...+2^n which adds up to $2^{2n-1} + 2^{n-1}$ or $O(4^n)$ substeps. Each substep takes time $O(n)$. The value at the end is broadcast to all nodes in each BB in $O(k)$ steps. Therefore, the time complexity for the second phase of the algorithm is $O(n4^n + k)$. Therefore, the overall time complexity of the algorithm is $O(n4^n + k)$. A hypercube with the same number of nodes would take time $O(2^n + k)$. In practical cases, the overhead due to missing links in RH's may not be significant.

CHAPTER 5

SORTING ON THE REDUCED HYPERCUBE (RH)

Sorting is one of the most common operations done on a computer. Many algorithms require sorted data as they are easier to manipulate than randomly ordered data. This section looks at an implementation of the sorting operation which can be done on the reduced hypercube parallel computer. Sorting is defined as the task of arranging an unordered collection of elements into monotonically increasing (or decreasing) order: without loss of generality, the increasing order is assumed. Specifically, let $S =< a_1, a_2, \ldots, a_p >$ be a sequence of p elements in arbitrary order; sorting transforms S into a monotonically increasing sequence $S' =< a'_1, a'_2, \ldots, a'_p >$, such that $a'_i \leq a'_j$ for all $1 \leq i < j \leq p$, and S' is a permutation of S.

The global order assumed by the algorithm is as follows: BB's are assumed ordered according to their 2^n-bit sequential addresses. The nodes inside a BB are assumed ordered according to their k-bit sequential addresses.

5.1 Sorting Algorithm

Let $N=2^{2^n+k}$ be the number of nodes in the $RH(k,n)$, with $k > n$. Let p be the number of data elements, where $p > N$. Initially each processor is assigned a block of p/N elements. The algorithm consists of three phases. In the first phase data elements in a BB are sorted. In the second phase data are sorted among BB's. In the third phase the sorted data are distributed within each BB.

5.1.1 Phase I: Sorting of Data within BB's

All the nodes sort the p/N elements internally using merge sort. All the k-cube BB's then sort their data using the bitonic sort algorithm [6]. To prepare for the second phase, each node in any particular BB sends its p/N sorted data elements to that node in the same BB whose address differs from its own address only in the subblock address bits, which are all zeros. This can be done in time $O(np/N)$ using the hypercube connections and the E-cube routing algorithm; it can take less time with wormhole routing. All nodes in the SB with all SB address bits zeros concatenate the incoming data elements to their own data elements in the increasing order of source SB addresses.

5.1.2 Phase II: Sorting between BB's

The algorithm takes advantage of the fact that a RH can be viewed as a hypercube of hypercubes (BB's), therefore bitonic sort [6] can be applied to the former hypercube. The algorithm does compare-exchange in dimension 0 first (this is the reason all data in a BB were moved to SB 0), then dimensions 1 and 0, in this order, then dimensions 2, 1 and 0, in this order, and so on. In each step the data elements are passed to the SB implementing connections in that respective dimension in all the BB's. Each physical processor involved in this phase can be viewed as 2^n virtual processors. Each time the $(k - n + 1)$-cube formed by the two SB's applies again the bitonic sort algorithm assuming a virtual $(n + 1)$-cube. At the end of the last step, the sorted elements in each BB are in the SB with address zero.

5.1.3 Phase III: Distribution of Sorted Values in BBs

The SB with address 0 in each BB will have the sorted sequence for the BB at the end of phase II. The sequence of $p/2^{2^n}$ elements in each such SB is divided into 2^n subsequences of consecutive elements for distribution to the other 2^n-1 SB's in the BB, so that global order is achieved. E-cube routing is used to distribute the subsequences.

5.2 Analysis of the Algorithm

Each node internally sorts its p/N data elements in time $O(p/N(log(p/N)))$ using merge sort. It takes time $O(p/N(log^2 2^k))$ or $O(p/N(k^2))$ for the values to be sorted in each BB using the bitonic sort algorithm. It takes $O(np/N)$ time for these sorted values to accumulate in the lowest addressed SB in each BB, because up to n dimensions may be traversed for each datum. In the second phase it takes $O((p/N)2^n)$ communication cycles to transfer sequences between neighboring SB's because $(p/N)2^n$ is the number of elements in each active processor. Bitonic sort in the $(k - n + 1)$-cubes formed by neighboring SB's in neighboring BB's takes time $O((p/N)log^2 2^{n+1})$ or $O((p/N)n^2)$ time. Lets denote the term $(p/N)2^n$ by the symbol β, and the term $(p/N)n^2$ by the symbol α. β denotes the time spent in communicating data elements between neighboring SB's, and α denotes the time spent in sorting data between SB's in neighboring BB's. The total asymptotic time complexity of the second phase of the algorithm is on the order of

$$\overbrace{\alpha}^{step\ 0:\ dim\ 0} + \overbrace{\beta+\alpha}^{step\ 1:\ dim\ 1} + \overbrace{\beta+\alpha}^{dim\ 0} + \overbrace{2\beta+\alpha}^{step\ 2:\ dim\ 2} + \overbrace{\beta+\alpha}^{dim\ 1} + \overbrace{\beta+\alpha}^{dim\ 0} + \cdots$$

where step i starts with the i^{th} dimension of the 2^n-cube of BB's.

This can be expressed by the following summation

$$O\left(\sum_{i=0}^{2^n-1}[(i+1)\alpha + 2i\beta]\right)$$

which simplifies to $O(4^n(\alpha+\beta))$ or $O((p/N)8^n)$, where $N = 2^{2^n+k}$. Thus it takes time $O(p/2^{k-n})$. The third phase takes time O(np/N), assuming that, higher priority is given for data transfers to SB's at larger distances. Thus, the total time complexity is $O(p/N(log(p/N))+(p/N)k^2+n(p/N)+p/2^{k-n})$. For small values of n (that is only k increases), this time is $O((p/2^k)k^2)$. In contrast, bitonic sort on the hypercube with the same number of nodes consumes time $O((p/N)log^2(2^{2^n+k})$, or $O(p/N(2^n+k)^2)$, or $O(p/N(4^n + k^2 + k2^n))$. For small values of n, this time is $O((p/2^k)k^2)$, thus achieving almost similar performance on both systems.

CHAPTER 6

CONCLUSIONS

The main focus of this thesis was to develop algorithms on the *RH* for operations which are very frequently used in many parallel algorithms. The algorithms which were developed are for data broadcasting and reduction, prefix operation, and sorting.

A one-to-one mapped binary tree was the basic structure used in broadcasting. The broadcasting operation was shown to be comparable to a similar operation on the hypercube. A many-to-one mapped binary tree was the basic structure used in the reduction operation. The reduction operation too was seen to be comparable to a similar operation on the hypercube. So these two operations should perform comparably well on the *RH*. The prefix computation and sorting algorithms also achieve comparable performance with the hypercube for systems with small number of missing links.

A *RH* has significantly lower VLSI complexity and comparable diameter and average internode distance compared to a regular hypercube with the same number of nodes [2]. The mapping on to *RH*'s of frequently used topologies, like the ring, the torus, and the binary tree have been shown to be efficient [17]. Algorithms for some frequently used operations in parallel algorithms were presented in this thesis. The results show that *RH*'s achieve good performance. Thus, it can be said that *RH*'s are a viable topology for building massively parallel hypercube-like systems.

REFERENCES

1. S.G. Ziavras, "On the Problem of Expanding Hypercube-Based Systems," *J. Parallel Distrib. Computing*, Sept. 1992, pp. 41-53

2. S.G. Ziavras, "RH: A Versatile Family of Reduced Hypercube Interconnection Networks," *To appear in IEEE Trans. Parallel Distrib. Systems.*

3. S.R. Deshpande and R.M. Jenevin, "Scalability of a Binary Tree on a Hypercube," *Proc Int Conf Parallel Proc*, IEEE Computer Society, Silver Spring, MD, Aug. 1986, pp. 661-668.

4. K. Ghose and K.R. Desai, "The HCN: A Versatile Interconnection Network based on Cubes," *Proc. Supercomputing '89*, IEEE Computer Society and ACM SIGARCH, Nov. 1989, pp. 426-435.

5. F.T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann Publ., San Mateo, CA, 1992, pp. 407-410.

6. G.C. Fox, M. Johnson, G. Lyzenga, S.W. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors: Vol 1*, Prentice Hall, Englewood Cliffs, NJ, 1988, pp. 327-343.

7. F.P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Comm. ACM* 24, May 1981, pp. 300-309.

8. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, Benjamin/Cummings Publ. Co., CA, 1994.

9. W.C. Athas and C.L. Seitz, "Multicomputers: Message-passing Concurrent Computers," *Computer*, 21(8), August 1988, pp. 9-24.

10. W.J. Dally and C.L. Seitz, "Deadlock-free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, May 1987, pp. 547-553.

11. T.H. Lai and W. White, W. "Mapping Pyramid Algorithms into Hypercubes," *J. Parallel Distrib. Comput.* 9(1990) pp. 42-54.

12. S.G. Ziavras and M.A. Siddiqui, "Pyramid Mappings onto Hypercubes for Computer Vision: Connection Machine Comparative Study," *Concurrency: Practice Experience, Vol. 5, No. 6, Sept. 1993*, pp. 471-489.

13. C.T. Ho and S.L. Johnson, "Spanning Balanced Trees in Boolean Cubes," *SIAM J. Sci. Stat. Comput.* July 1989, pp. 607-630.

14. T.F. Chan and Y. Saad. "Multigrid Algorithms on the Hypercube Multiprocessor," *IEEE Trans. Computers* C-35, Aug 1988, pp. 969-977.

15. A.Y. Wu, "Embedding of Tree Networks into Hypercubes," *J. Parallel Distrib. Comput.*, Aug 1985, pp. 238-249.

16. S.L. Johnsson, "Communication Efficient Basic Linear Algebra Computation on Hypercube Architectures," *J. Parallel Distrib. Comput.*, Apr 1987, pp. 133-172.

17. S.G. Ziavras and M.A. Sideras, "Facilitating High-Performance Image Analysis on Reduced Hypercube (RH) Parallel Computers," *Proc.* 3rd *International Workshop Parallel Image Analysis: Theory Applications*, College Park, MD, June 7-9, 1994, pp. 263-291.

18. H.P. Katseff, "Incomplete Hypercubes," *IEEE Trans. Computers*, May 1988, pp. 604-608.

19. S.G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1989.

20. R. Sauya and G. Birtwistle, editors, *VLSI and Parallel Computation*, Morgan Kaufman, San Mateo, CA, 1990.

21. K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw Hill, New York, NY, 1993.

22. C.P. Kruskal, L. Rudolph, M. Snir, "The Power of Parallel Prefix," *IEEE Trans. Computers*, Oct 1985, pp. 965-968.

23. D. Nassimi and S. Sahni, "Bitonic Sort on a Mesh Connected Parallel Computer," *IEEE Trans. Computers*, February 1979.

24. G.M. Baudet, D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers," *IEEE Trans. Computers*, Jan 1978, pp. 84-87.

25. C.P. Kruskal, "Searching, Merging and Sorting in Parallel Computations," *IEEE Trans. Computers*, Oct 1983, pp. 942-946.